

# A structural and nominal syntax for diagrams

Dan R. Ghica, University of Birmingham  
Aliaume Lopez, ENS Cachan, Université Paris-Saclay

February 7, 2017

## Abstract

The correspondence between various monoidal categories and graphical languages of diagrams has been studied extensively, leading to applications in quantum computing and communication, systems theory, circuit design and more. From the categorical perspectives, diagrams can be specified using (name-free) combinators which enjoy elegant equational properties. However, established notations for diagrammatic structures, such as hardware description languages (VHDL, VERILOG) or graph languages (DOT), use a different style, which is flat, relational, and reliant on extensive use of names (labels). Such languages are not known to enjoy nice syntactic equational properties. However, since they make it relatively easy to specify (and modify) arbitrary diagrammatic structures they are much more popular than the combinator-style specifications. This parallels the situation in programming languages, where combinator-based languages (e.g. APL) proved to be far less popular than languages with variables. In this paper we show how the two approaches to diagram syntax can be reconciled and unified in a way that does not change the original categorical semantics and the existing (categorical) equational theory. We also give sound and complete equational theories for the combined syntax.

## 1 Specifying graphs

Graphs and their visual representations (diagrams) are a very appealing way of describing many kinds of systems, in particular circuits. Work originated initially in the study of quantum computation [1] has exploited the connection between various classes of graphs and monoidal categories, going back to the seminal work of Joyal and Street [12], to add a layer of structure which makes reasoning about and with diagrams not just intuitive but also mathematically rigorous. Subsequently this connection was extended in many surprising and interesting directions, from computational linguistics [15], to modelling signal flow [3] and synchronous [8] or asynchronous [7] circuits. New automated reasoning tools based on diagrams, rather than the usual linear algebraic syntax, are a particularly exciting development [18]. This convenient and elegant interplay between the dual categorical and diagrammatic methods are by now a mature and rich area of research [16].

Although these developments convincingly establish the usefulness of categorical diagrammatics in *reasoning* about many kinds of systems, we note that little has been suggested in terms of a workable syntax which is conceptually compatible with it, but also with the more conventional notations which tend to be flat and relational. Examples of the latter are hardware description languages such as VERILOG and VHDL, or graph languages such as DOT<sup>1</sup>. Indeed, in the literature the categorical combinators are usually taken as an implicit diagram syntax. Whereas such a syntax is expressive enough to describe the desired classes of graphs, it is often not as convenient as the alternatives. Although categorical combinators can be elegant and succinct in certain situation, *point-free* languages of combinators generally hold little appeal as concrete syntax (e.g. APL).

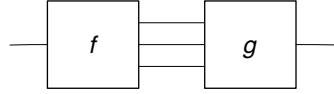
The established diagram syntax, mentioned earlier, is profoundly different. It is essentially structure-free, a “flat” relational description of the graph. Whereas the categorical combinators are name-free, conventional syntax uses an abundance of names. In hardware languages, for example, every component, every wire, and every port requires a unique name (up to some scoping and visibility rules). Although these languages are widely used, they are broadly considered both

---

<sup>1</sup><http://www.graphviz.org/>

inelegant and unwieldy, and purely structural alternatives have been proposed [2]. The large number of names imposes a cognitive burden on the programmer and makes refactoring cumbersome. Reasoning at the level of syntax, equationally, is impossible and, as a consequence, operational semantics of languages which such a syntax did not (until recently [9]) exist.

Before diving into technicalities let us consider a simple motivating example. Consider the diagram below, consisting of two components  $f$  and  $g$ , the first with one input and three outputs and second with three inputs and one output, connected in the obvious way:



In the categorical, combinator-style, specification such a diagram would be succinctly written as the *composition*  $f;g : 1 \rightarrow 1$ , of  $f : 1 \rightarrow 3$  and  $g : 3 \rightarrow 1$ . An idealised VHDL-style description would look far more verbose:

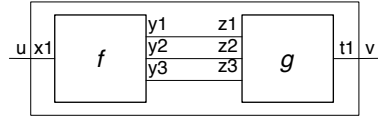
```

module fg(input u; output v)
begin
component f(input x1; output y1, y2, y3);
component g(input z1, z2, z3; output t1);
wire y1, z1;
wire y2, z2;
wire y3, z3;
wire u, x1;
wire t1, v;
end

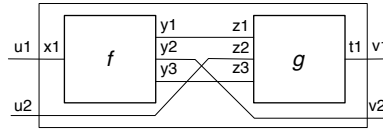
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

This is the diagram annotated with all the names used above:



In highly structured diagrams the categorical notation is concise and elegant. However, realistic circuits often mix structural elements with arbitrary connectors. Consider a variation of the circuit above:



The flat description can be quite easily adjusted to cope with arbitrary connector reassignments:

```

1 module fg(input u1, u2; output v1, v2)
2 begin
3 component f(input x1; output y1, y2, y3);
4 component g(input z1, z2, z3; output t1);
5 wire y1, z1;
6 wire y3, z3;
7 wire y2, v2;
8 wire u2, z2;
9 wire u1, x1;
10 wire t1, v1;
11 end

```

The categorical style description is now more intricate, requiring a mix of sequential (;) and parallel (*tensor*,  $\otimes$ ) composition along with the use of *structural* combinators such as the single connector (*identity*,  $i$ ) and crossing connector (*symmetry*,  $\gamma$ ):

$$(f \otimes i); (i \otimes \gamma \otimes i); (i \otimes i \otimes \gamma); (i \otimes \gamma \otimes i); (g \otimes i).$$

The combinator-style variable-free syntax is appealing for highly structured diagrams and awkward for arbitrary ones, whereas conversely, the flat and unstructured syntax is appealing for unstructured and unnecessarily verbose for structured graphs.

In this paper we present a syntax which gives the best of both worlds, conservatively extending the categorical syntax with a powerful and flexible notion of variable which will allow the exploitation of underlying structure wherever it exists, but also arbitrary connection whenever needed. Our contributions are as follows:

1. Preserving and extending the categorical equations, thus not invalidating formal reasoning principles.
2. Eliminating the need for using structural combinators (such as identity, symmetry, trace, etc.) wherever desired.
3. Preserving and conserving the expressiveness of the underlying categorical framework, offering only convenience and not semantic extensions.
4. Allowing the specification of arbitrary graphs in an elegant and intuitive way.
5. Reasoning equationally in the new syntax.

## 2 Uniflow diagrams

A PROP (abbreviation of *products and permutations*) is a strict symmetric monoidal category where every object is a natural number [10]. We give a graph semantics of PROPs based on Kissinger’s *framed point graphs* [13].

Let a labelled directed acyclic graph (DAG) be a DAG  $(V, E)$  equipped with a partial injection  $f : V \rightarrow L$  and a relation  $E \subseteq V^2$  such that the transitive and reflexive closure of  $E$  is a partial order on  $V$ . Let a labelled interfaced DAG (LIDAG) be a labelled DAG with two distinguished lists of unlabelled nodes representing the “input” and “output” ports. Unlabelled nodes are called *wire nodes*, and edges connecting them are called *wires*.

A wire homeomorphism [13, Sec. 5.2.1] is any insertion or removal of wire nodes along wires which does not otherwise change the shape of the graph. The rewrite rules are:

$$\bullet \rightarrow \bullet \rightarrow \bullet \quad \leftarrow \rightleftarrows \rightarrow \quad \bullet \rightarrow \bullet \rightarrow \bullet$$

Two LIDAGs are considered to be equivalent if they are graph isomorphic up to renaming vertices and wire homeomorphisms. The quotienting of LIDAGs by this equivalence gives us *framed point DAGs* [13, Def. 5.3.1], which we will call *uniflow diagrams*<sup>2</sup>. In uniflow diagrams the arrows all point, conventionally, left-to-right.

Let us introduce a syntax for describing uniflow diagram

$$M ::= k \mid i \mid \gamma \mid M; M \mid M \otimes M \mid x \mid \overline{xy}.M, \tag{1}$$

where  $k \in K$  are the constants and  $x$  variable identifiers.

The language is essentially that of symmetrical monoidal categorical combinators (identity, symmetry, composition, tensor) extended with variables and a binding construct we read as *link  $x$  and  $y$  in  $M$* .

We give this language an associated type system, where judgments are of the form

$$\Gamma \mid \Delta \vdash M : m \rightarrow n \mid R.$$

<sup>2</sup>This is by contrast with *biflow diagrams* described in the following section.

$$\begin{array}{c}
\frac{}{x \mid - \vdash x : 0 \rightarrow 1 \mid x \leq x} \quad \frac{}{- \mid x \vdash x : 1 \rightarrow 0 \mid x \leq x} \quad \frac{}{- \mid - \vdash i : 1 \rightarrow 1 \mid \emptyset} \\
\frac{}{- \mid - \vdash \gamma : 2 \rightarrow 2 \mid \emptyset} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \mid \leq \quad \Gamma' \mid \Delta' \vdash N : n_1 \rightarrow n_2 \mid \leq'}{\Gamma \uplus \Gamma' \mid \Delta \uplus \Delta' \vdash M; N : m_1 \rightarrow n_2 \mid [\leq \cup \leq' \cup ((\Gamma \uplus \Delta) \times (\Gamma' \uplus \Delta'))]} m_2 = n_1 \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \mid \leq \quad \Gamma' \mid \Delta' \vdash N : n_1 \rightarrow n_2 \mid \leq'}{\Gamma \uplus \Gamma' \mid \Delta \uplus \Delta' \vdash M \otimes N : m_1 \rightarrow n_2 \mid [\leq \cup \leq']} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \mid \leq \quad [\leq \cup \{(x, y)\}] \text{ is a partial order}}{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \mid [\leq \cup \{(x, y)\}]} x, y \in \Gamma \uplus \Delta \\
\frac{\Gamma, x \mid \Delta, y \vdash M : m_1 \rightarrow m_2 \mid \leq \quad x \leq y}{\Gamma \mid \Delta \vdash \overline{xy}.M : m_1 \rightarrow m_2 \mid \leq \setminus \{x, y\}}
\end{array}$$

Figure 1: Uniflow diagrams typing rules

$$\begin{aligned}
\llbracket x \mid - \vdash x : 0 \rightarrow 1 \mid x \leq x \rrbracket &= (\{a, b\}, \emptyset, \{b\}, \{(a, b)\}, \{(a, x)\}) \asymp \{(a, a)\} = \begin{array}{c} \boxed{\phantom{a}} \quad \overset{x}{\bullet} \longrightarrow \boxed{\bullet} \end{array} \\
\llbracket - \mid x \vdash x : 1 \rightarrow 0 \mid x \leq x \rrbracket &= (\{a, b\}, \{a\}, \emptyset, \{(a, b)\}, \{(b, x)\}) \asymp \{(b, b)\} = \begin{array}{c} \boxed{\bullet} \longrightarrow \overset{x}{\bullet} \quad \boxed{\phantom{a}} \end{array} \\
\llbracket - \mid - \vdash i : 1 \rightarrow 1 \mid \emptyset \rrbracket &= (\{a, b\}, \{a\}, \{b\}, \{(a, b)\}, \emptyset) \asymp \emptyset = \begin{array}{c} \boxed{\bullet} \longrightarrow \boxed{\bullet} \end{array} \\
\llbracket - \mid - \vdash \gamma : 2 \rightarrow 2 \mid \emptyset \rrbracket &= (\{a, a', b, b'\}, \{a, a'\}, \{b, b'\}, \{(a, b'), (a', b)\}, \emptyset) \asymp \emptyset = \begin{array}{c} \boxed{\bullet \bullet} \longrightarrow \boxed{\bullet \bullet} \end{array}
\end{aligned}$$

Figure 2: Interpreting structural combinators

$\Gamma$  is a set of *input variables*,  $\Delta$  of *output variables* and  $R$  a partial order on their (disjoint) union. We call this order the *anchor* of the diagram and we use it to prevent cycles in the graph. The type  $m \rightarrow n$  represents a uniflow diagram with  $m$  (unlabelled) inputs and  $n$  (unlabelled) outputs. Given a relation  $R$  we denote by  $[R]$  its transitive and reflexive closure. For any set  $S$ , we define

$$R \setminus S = \{(x, y) \in R \mid x, y \notin S\}.$$

The typing rules are given in Fig. 1.

In this section we will give a *concrete diagrammatic semantics*, rather than categorical, for the syntax in Eq. 1. However, the correspondence between the diagram and monoidal categories will be useful to render many proofs immediate.

In general we may write  $\Gamma \mid \Delta \vdash M$  if  $\Gamma \mid \Delta \vdash M : m \rightarrow n \mid R$  for some  $m, n \in \mathbb{N}$  and some partial order  $R$ .

The uniflow semantics is given by induction on the typing derivation. The meaning of a judgement  $\Gamma \mid \Delta \vdash M : m \rightarrow n \mid R$  is a uniflow diagram  $(V, I, O, E, f : V \rightarrow K \uplus \mathbb{A})$  such that  $\mathbb{A} \supseteq \Gamma, \Delta$  is a set of names and the relation  $R$  is a partial order on the vertices labelled by variables  $\Gamma \uplus \Delta$  which is compatible with  $E$ , i.e. if  $(x, y) \in R$  then there exist unique vertices  $u, v$  such that  $f(u) = x, f(v) = y$  and *there is no path* in  $E$  from  $v$  back to  $u$ . We write this as

$$[\Gamma \mid \Delta \vdash M : m \rightarrow n \mid R] = (V, I, O, E, f) \asymp R.$$

We indicate the input and output lists, hide the nodes' identities, but display their labels, if any. The structural combinators are interpreted as in Fig. 2.

The two interpretations of the two forms of composition is given below. Let

$$[\Gamma_i \mid \Delta_i \vdash M_i : m_i \rightarrow n_i \mid R_i] = (V_i, I_i, O_i, E_i, f_i) \asymp R_i$$



**Lemma 1** (Soundness). *Any well typed term denotes a valid uniflow diagram.*

*Proof.* Immediate, by induction on derivation.  $\square$

The useful connection between diagrams and monoidal categories is given by this proposition:

**Proposition 2.** *Given a term  $\Gamma \mid \Delta \vdash M : m \rightarrow n$ , the diagram  $\llbracket \Gamma \mid \Delta \vdash M : m \rightarrow n \rrbracket$  is a morphism in the strict free symmetric monoidal category (SMC) over signature  $K \cup \Gamma \cup \Delta$  of constants and (uninterpreted) variables.*

**An important note:** we cannot deem our semantics to be categorical, even though each diagram is a morphism in this SMC, with sequential composition as morphism composition and tensorial composition as monoidal tensor. The reason is that *link* is not interpreted by a categorical construct in the SMC. A proper categorical semantics will be discussed in the concluding section but left as further work. Although a categorical semantics would be certainly desirable, the concrete diagrammatic interpretation is enough for soundness and completeness. And even in the absence of a categorical semantics, the connection between diagrams and monoidal categories will prove useful in enabling the proof of definability!

An immediate consequence of Prop. 2 is that all derivations of a well typed term produce the same uniflow diagram, possibly with a different anchoring relation.

**Lemma 3.** *The type system of uniflow diagrams is coherent up to the anchoring relation.*

Note that different derivations may require different anchoring relations, even though the underlying diagrams are equal as frame point dags.

**Lemma 4** (Definability). *Any uniflow diagram is definable just in terms of composition, tensor, link.*

*Proof.* We introduce exactly two link nodes on each connector (a wire homeomorphism) and we label them with fresh variable names. Suppose there are  $j$  wires and  $k$  boxes,  $m$  inputs and  $n$  outputs. Let us denote by  $u_i$  where  $i = 1, n$  the labels associated with input connectors and by  $v_i$  where  $i = 1, m$  the labels associated with the output connectors. Let  $y_{i,j}$  (respectively  $z_{i,j}$ ) be the labels of the  $j$ th input (respectively output) for each constant occurrence  $k_i$ . Let

$$\begin{aligned} \vec{u} &= \bigotimes_{i=1,n} u_i, & \vec{v} &= \bigotimes_{i=1,m} v_i, \\ \vec{y}_i &= \bigotimes_{j=1, \text{dom}(k_i)} y_{i,j}, & \vec{z}_i &= \bigotimes_{j=1, \text{codom}(k_i)} z_{i,j}. \end{aligned}$$

The term is:

$$M = \overline{x_1 x'_1} \cdots \overline{x_n x'_n} . \vec{u}; \left( \bigotimes_{i=1,k} \vec{y}_i; f_i; \vec{z}_i \right); \vec{v}.$$

It is straightforward to show that this is indeed denotes the desired diagram. We only need to prove that this is a well typed term. The subterm  $\bigotimes_{i=1,k} \vec{y}_i; f_i; \vec{z}_i$  is clearly well typed. Moreover, we can always extend the order with any  $x_i \leq x'_i$  before linking without creating cycles because the starting graph is a DAG.  $\square$

Note that the construction in the proof gives precisely the flat nominal notation used by conventional graph languages.

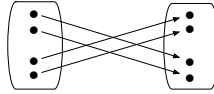
**Theorem 5.** *For any term  $M$  there exist terms  $\tilde{M}$  and  $\hat{M}$  such that  $\tilde{M}$  is link-free and  $\hat{M}$  is free of structural combinators (identity, symmetry) and  $\llbracket M \rrbracket = \llbracket \tilde{M} \rrbracket = \llbracket \hat{M} \rrbracket$ .*

*Proof.* Given a term  $\Gamma \mid \Delta \vdash M : m \rightarrow n$  we construct the uniflow diagram  $\llbracket M \rrbracket$ . The uniflow diagrams is a morphism in the free strict symmetric monoidal category over the signature  $K$  extended with variables  $\Gamma, \Delta$  (Prop. 2) so the existence of  $\tilde{M}$  is immediate. The existence of  $\hat{M}$  is given by Lem. 4.  $\square$

$\Gamma \mid \Delta \vdash (M_1; M_2); M_3 \equiv M_1; (M_2; M_3)$	associativity of composition	(2)
$\Gamma \mid \Delta \vdash m; M \equiv M; n \equiv M$	identity	(3)
$\Gamma \mid \Delta \vdash (M; M') \otimes (N; N') \equiv (M \otimes N); (M' \otimes N')$	tensor functoriality	(4)
$\Gamma \mid \Delta \vdash (M_1 \otimes M_2) \otimes M_3 \equiv M_1 \otimes (M_2 \otimes M_3)$	strictness	(5)
$\Gamma \mid \Delta \vdash M \otimes 0 \equiv 0 \otimes M \equiv M$	strictness	(6)
$\Gamma \mid \Delta \vdash (M_1 \otimes M_2); \gamma_{n_1, n_2} \equiv \gamma_{m_2, m_1}; (M_2 \otimes M_1)$	symmetry	(7)
$\vdash \gamma_{0,1} \equiv \gamma_{1,0} \equiv 1$	strict symmetry	(8)
$\vdash \gamma_{m, n+p} \equiv (\gamma_{m, n} \otimes p); (n \otimes \gamma_{m, p})$	strict symmetry	(9)
$\Gamma \mid \Delta \vdash \overline{xy}.M \equiv \overline{x'y'}.M\{x'y'/xy\} \quad x', y' \text{ fresh}$	alpha-equivalence	(10)
$\Gamma \mid \Delta \vdash \overline{xy}.(M; N) \equiv (\overline{xy}.M); N \quad x, y \notin \text{fv}(N)$	scope extrusion	(11)
$\Gamma \mid \Delta \vdash \overline{xy}.(M; N) \equiv (\overline{xy}.M); N \quad x, y \notin \text{fv}(N)$	scope extrusion	(12)
$\Gamma \mid \Delta \vdash \overline{xy}.(M; N) \equiv M; \overline{xy}.N, \quad x, y \notin \text{fv}(M)$	scope extrusion	(13)
$\Gamma \mid \Delta \vdash \overline{xy}.(M \otimes N) \equiv M \otimes \overline{xy}.N, \quad x, y \notin \text{fv}(M)$	scope extrusion	(14)
$\vdash \overline{xy}.x; y \equiv 1$	link identity	(15)

Figure 4: Equational theory of uniflow diagrams

**Example 1.** *The term language only has symmetry at the unit type 1. Symmetry at other types, such as  $\gamma_{2,2} : 4 \rightarrow 4$ :*



can be defined in purely combinatorial or purely nominal style as follows:

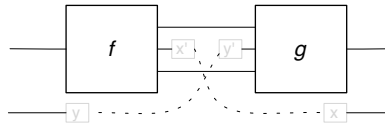
$$\gamma_{2,2} = \overline{aa'}. \overline{bb'}. \overline{cc'}. \overline{dd'}. a \otimes b \otimes c \otimes d \otimes c' \otimes d' \otimes a' \otimes b' = (i \otimes \gamma \otimes i); (\gamma \otimes \gamma); (i \otimes \gamma \otimes i).$$

In fact any  $\gamma_{m,n}$  for  $m, n \neq 0$  can be defined from  $\gamma$ . Similarly, identity at any type which is not zero can be defined from the identity at 1.

**Example 2.** *We can now revisit our motivating example. The most succinct description mixes variables and structural connectors:*

$$\overline{x'x}. \overline{yy'}. (f \otimes y); (i \otimes x' \otimes y' \otimes i); (g \otimes x),$$

visualised as



## 2.1 Equational theory for links

It can be easily checked that all equational properties of the underlying PROP are carried over to the syntax. We interpret  $\Gamma \mid \Delta \vdash M \equiv N$  as the equality of the uniflow diagrams represented by  $\llbracket \Gamma \mid \Delta \vdash M \rrbracket$  and  $\llbracket \Gamma \mid \Delta \vdash N \rrbracket$  (ignoring the anchoring order). The equations are now parameterised by the set of free variables, as in Fig. 4, assuming integers  $m, n$ , etc. are chosen so that the terms are well-typed.

Although from a circuit-description point of view the equi-expressiveness of the nominal and structural notations is the essential property, it is interesting to also examine the equational properties of links. Since  $\overline{xy}.M$  is a binding construct, we expect alpha-equivalence to hold, which it

$$\begin{array}{c}
\frac{}{x \mid - \vdash x : 0 \rightarrow 1} \quad \frac{}{- \mid x, \vdash x : 1 \rightarrow 0} \quad \frac{}{- \mid - \vdash i : 1 \rightarrow 1} \quad \frac{}{- \mid - \vdash \gamma : 2 \rightarrow 2} \\
\\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma' \mid \Delta' \vdash N : n_1 \rightarrow n_2}{\Gamma \uplus \Gamma' \mid \Delta \uplus \Delta' \vdash M; N : m_1 \rightarrow n_2} \quad m_2 = n_1 \\
\\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma' \mid \Delta' \vdash N : n_1 \rightarrow n_2}{\Gamma \uplus \Gamma' \mid \Delta \uplus \Delta' \vdash M \otimes N : m_1 \rightarrow n_2} \\
\\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2}{\Gamma \mid \Delta \vdash \text{Tr}^i(M) : m_1 - i \rightarrow m_2 - i} \quad \frac{\Gamma, x \mid \Delta, y \vdash M : m_1 \rightarrow m_2}{\Gamma \mid \Delta \vdash \overline{xy}.M : m_1 \rightarrow m_2}
\end{array}$$

Figure 5: Type system for biflow diagrams

does via the new axiom 10 in Fig. 4, where renaming is defined in the obvious way. The set of free variables  $fv(M)$  is also defined in the obvious way.

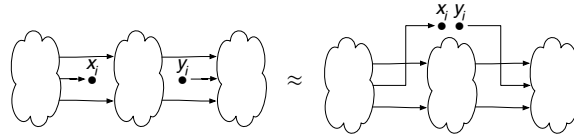
The new equations are for scope extrusion (Eqn. 11–14) and identity (Eq. 15 in Fig. 4).

**Theorem 6.** *The equational theory of uniflow diagrams is sound and complete.*

*Proof.* The soundness is immediate by checking each equation in the diagrammatic semantics.

For completeness, consider two terms such that  $\llbracket \Gamma \mid \Delta \vdash M \rrbracket = \llbracket \Gamma \mid \Delta \vdash N \rrbracket$ . We first rename all bound variables using alpha-equivalence, then, using Eqns. 11 and 13 we move all binders into global scope. In case the resulting terms are link-free the equation  $\Gamma \mid \Delta \vdash M \equiv N$  is provable because the terms denote morphism in the free strict symmetric monoidal category over signature  $K$  extended with the uninterpreted variables  $\Gamma, \Delta$  (Prop. 2).

Suppose that one of the terms has an outermost link, so it has form  $\Gamma \mid \Delta \vdash \overline{xy}.M$ . We reason diagrammatically about term  $\Gamma, x \mid \Delta, y \vdash M$ , which is a morphism in the free SMC over  $K$  extended with  $\Gamma \uplus \Delta \uplus \{x, y\}$ . We can always redraw the diagram to an isomorphic diagram in which the nodes labelled by  $x$  and  $y$  are adjacent. This is informally illustrated below:



This diagrammatic equivalence can be proved in the equational theory of the uniflow diagrams, noting that  $\gamma_{m,n}$  and the identities at  $m$  can be defined in terms of  $\gamma$  and  $i$ . Moreover, the diagram on the right will correspond to a term which has subterm  $x; y$ , with variables  $x, y$  not occurring anywhere else. So we can apply Eqns. 11 and 13 repeatedly until the link binder only binds this subterm,  $\overline{xy}.x; y$ . Then we use Eqn. 15 to replace this subterm with the identity. We repeat the process until all binders are eliminated.  $\square$

### 3 Biflow diagrams

Let us consider now strict symmetric *traced* monoidal categories and their associated diagrammatic language, *biflow diagrams* [5]. Biflow diagrams are labelled directed graphs with vertices  $V$  and edges  $E$ , equipped with a partial injection  $f : V \rightarrow L$  from vertices to labels. Biflow diagrams are labelled directed graphs with interfaces  $I, O$ , lists of unlabelled ports, equivalent up to vertex renaming and wire homeomorphism. In other words, biflow diagrams are uniflow diagrams minus the anchoring relation, similar to Kissinger's *frame point graphs*. Cycles are allowed. As in the case of uniflow diagrams, connecting arbitrary points in the diagram is potentially awkward in the language of categorical combinators, a motivation for the *link* construct.

The increased expressivity of the diagrammatic language is reflected into a relaxation of the type system. The type judgements are  $\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2$ , similar to the uniflow case but without an order on the free variables. The rules are also similar, minus the anchoring order, but new rules for *trace* (Fig. 5). If the trace is over 1 we omit the superscript. Note that traces with



$$\begin{array}{lll}
\Gamma \mid \Delta \vdash \text{Tr}^m((g \otimes n); f) \equiv g; \text{Tr}^m(f) & \text{left-tightening} & (16) \\
\Gamma \mid \Delta \vdash \text{Tr}^m(f; (g \otimes n)) \equiv \text{Tr}^m(f); g & \text{right-tightening} & (17) \\
\Gamma \mid \Delta \vdash \text{Tr}^m(f; (n \otimes g)) \equiv \text{Tr}^p((q \otimes g); f) & \text{sliding} & (18) \\
\Gamma \mid \Delta \vdash \text{Tr}^0(f) \equiv f & \text{vanishing} & (19) \\
\Gamma \mid \Delta \vdash \text{Tr}^{m+n}(f) \equiv \text{Tr}^m(\text{Tr}^n(f)) & \text{vanishing} & (20) \\
\Gamma \mid \Delta \vdash \text{Tr}^m(n \otimes s) \equiv n \otimes \text{Tr}^m(f) & \text{superposing} & (21) \\
\vdash \text{Tr}^1(\gamma) \equiv 1 & \text{yanking} & (22) \\
\Gamma \mid \Delta \vdash \overline{xy}. \text{Tr}^i(M) \equiv \text{Tr}^i(\overline{xy}.M), \quad i \in \{0, 1\}. & \text{scope extrusion} & (23)
\end{array}$$

Figure 6: Additional equations for biflow diagrams

$i > 1$  can be constructed out of unit traces.

The diagrammatic interpretations are the same as in the previous section, omitting the anchoring order on vertices. The interpretation of the trace operator is the standard one, a *feedback* edge between the top input and output ports. Soundness holds, immediately. The other key properties have proofs similar to those in the previous section, made simpler by the absence of the partial order on graph vertices.

**Lemma 7** (Biflow definability). *Any biflow diagram is definable in terms of composition, tensor, link.*

*Proof.* The proof is the same construction as in Lem. 4, leading to the new term:

$$\overline{x_1 x'_1} \cdots \overline{x_n x'_n} \cdot \bigotimes_{i=1,k} \vec{y}_i f_i \vec{z}_i.$$

In this case the derivation of the term is obviously possible, since the rules are strictly more relaxed than in the case of DAGs.  $\square$

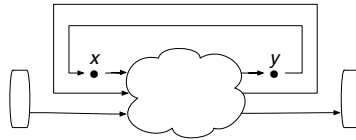
**Theorem 8.** *For any term  $M$  in the language of biflow diagrams, there are forms  $\tilde{M}$  and  $\hat{M}$  such that  $\tilde{M}$  is link-free and  $\hat{M}$  is free of structural combinators (identity, symmetry, trace) and  $\llbracket M \rrbracket = \llbracket \tilde{M} \rrbracket = \llbracket \hat{M} \rrbracket$ .*

*Proof.* The proof is immediate from the definability of biflow diagrams (Lem. 7).  $\square$

### 3.1 Equational theory of biflow diagrams

In the equational theory we inherit all the equations from the previous section, plus the trace equations [11]. All the additional equations are given in Fig. 6. Surprisingly, we do not need new equations for link, except for scope extrusion in the presence of trace (Eqn. 23).

Other relations between trace and link can be derived in the existing equational theory. The proposition below is clearly sound in the model, corresponding (informally) to two alternative descriptions of the same diagram, for example



The proposition can also be derived equationally.

**Proposition 9.**

$$\Gamma \mid \Delta \vdash \text{Tr}(M) \equiv \overline{yx}.(x \otimes m); M; (y \otimes n) \quad (24)$$

*Proof.*

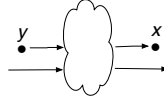
$$\begin{aligned}
& \overline{y\bar{x}}.(x \otimes m); M; (y \otimes n) \\
&= \overline{y\bar{x}}.\text{Tr}^0((x \otimes m); M; (y \otimes n)) && \text{(vanishing)} \\
&= \overline{y\bar{x}}.\text{Tr}(m; M; (y; x \otimes n)) && \text{(sliding)} \\
&= \overline{y\bar{x}}.\text{Tr}(M; (y; x \otimes n)) && \text{(identity)} \\
&= \text{Tr}(M; (\overline{y\bar{x}}.(y; x) \otimes n)) && \text{(link scope)} \\
&= \text{Tr}(M; (1 \otimes n)) && \text{(link)} \\
&= \text{Tr}(M) && \text{(identity)}
\end{aligned}$$

□

**Theorem 10.** *The equational theory of biflow diagram is sound and complete.*

*Proof.* The proof is the same as for Thm. 6, except that the ambient diagrammatic theory is now different. Diagram manipulation required to create subterms  $x; y$  si now allowed to involve the use of trace, so we need not assume an ordering on the variables. The additional scope extrusion rule allows all the binders to be moved equationally to outermost scope, so that all terms have shape  $\Gamma \mid \Delta \vdash \overline{x_1 y_1} \dots \overline{x_n y_n}. M'$  with  $M'$  binder-free.

Suppose that a term has binder  $\Gamma \mid \Delta \vdash \overline{xy}. M$ . We reason diagrammatically about  $\Gamma, x \mid \Delta, y \vdash M$ . Using the equational theory for TMCs we can redraw the diagram so that  $y$  is level with the input interface and  $x$  level with the output interface:



This means that the term has form

$$\Gamma, x \mid \Delta, y \vdash M \equiv (y \otimes m); M'(x \otimes n)$$

for some natural numbers  $m, n$  and term  $M'$ , provable in the equational theory of the TMC. But using Prop. 9 we can now replace the binder with a trace, equationally. Therefor, all binders can be replaced with traces *equationally*. The resulting binder-free terms, since they denote equal diagrams, must be also equationally equal in the SMT theory. □

## 4 Monoid and comonoid structures

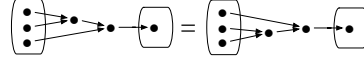
In the specification of diagrams corresponding to circuits and systems two kinds of structural components are used quite extensively: the splitting of two connectors and the joining of two connectors. A *monoid*, in this particular diagrammatic context is a pair of constant morphisms  $(\phi, \eta)$  where  $\phi : 2 \rightarrow 1$  is the *multiplication* and  $\eta : 1 \rightarrow 0$  the *unit*, interpreted as the following biflow diagrams corresponding to “joining two connectors” and, respectively, to a “dangling output” connector:

$$\llbracket \phi : 2 \rightarrow 1 \rrbracket = \boxed{\begin{array}{c} \bullet \\ \bullet \end{array}} \rightarrow \bullet \rightarrow \boxed{\bullet} \quad \llbracket \eta : 1 \rightarrow 0 \rrbracket = \bullet \rightarrow \boxed{\bullet}$$

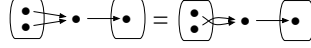
Conversely, a *co-monoid* consist of a *co-multiplication*  $\psi : 1 \rightarrow 2$  corresponding to “splitting” two connectors and a *co-unit*  $\epsilon : 0 \rightarrow 1$ , a “dangling input”.

$$\llbracket \psi : 1 \rightarrow 2 \rrbracket = \boxed{\bullet} \rightarrow \bullet \leftarrow \boxed{\begin{array}{c} \bullet \\ \bullet \end{array}} \quad \llbracket \epsilon : 1 \rightarrow 0 \rrbracket = \boxed{\bullet} \rightarrow \bullet$$

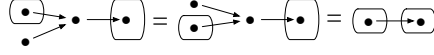
The monoid should be associative  $(\phi \otimes 1); \phi = (1 \otimes \phi); \phi$ , represented as:



and commutative  $\phi = \gamma; \phi$ , represented as



and have a unit law  $(1 \otimes \eta); \phi = (\eta \otimes 1); \phi = 1$ , represented as

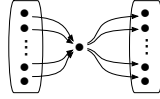


The co-monoid should be co-associative, co-commutative and have a co-unit, with the equations and diagrams the converse of the above.

Other laws that sometime make sense for diagram connectors are *Frobenius*  $((\phi \otimes 1); (1 \otimes \psi) = \phi; \psi)$  and the *special* law  $(\psi; \phi = 1)$ . Diagrammatically they are:



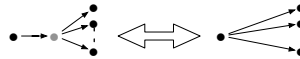
If all these equations are satisfied, any structure constructed out of the multiplication, co-multiplication, unit and co-unit can be reduced to a canonical form called a *spider diagram* [4]. Informally, all such constructions will denote a diagram of shape equivalent to



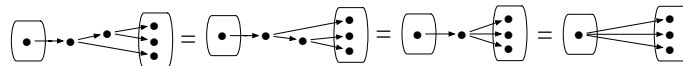
Precisely how these equations are chosen and how they are incorporated into the underlying graph model presents many choices that must be dealt with on a case-by-case basis, depending on the intending applications. In the next two sub-sections we will consider two interesting scenarios, but many more variations are possible.

#### 4.1 Comonoid structure

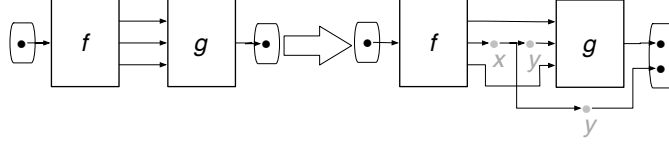
One natural way to add a comonoid structure to a biflow diagram is by adding constant morphisms  $\psi : 1 \rightarrow 2$  and  $\epsilon : 1 \rightarrow 0$  with the semantics of the previous section. The wire-homeomorphism rewrite rules, which consist of the removal of all possible unlabelled nodes (or, conversely, the insertion of spurious unlabelled nodes) carry over in the obvious way to the new setting. Note that in the absence of the monoid structure all wire nodes, by construction, have exactly one incoming and one outgoing edge. With the monoid structure in place, every wire node has exactly one incoming edge and zero, one or more outgoing edges and wire homeomorphisms need to re-assign the target of the edge. The rewrite rule for removing or inserting a wire node is:



The associativity, co-commutativity and co-unit laws now come then directly from the new wire homeomorphism. For example  $\psi; (\psi \otimes 1) = \psi; (1 \otimes \psi)$  is:



From the point of view of syntax, the presence of new structural connectors creates new opportunities for expressiveness but also for complications. In the case of circuit specifications, for example, a very common modification of a circuit that a designer might want to make is to identify an arbitrary point in the circuit as an observable output, for debugging purposes. For example, in the example circuit we used for motivation, the designer might want to observe the second output of  $f$ :



We recall that the circuit on the left was simply  $f;g$ . One way to write the circuit on the right is

$$f; (1 \otimes \psi \otimes 1); (1 \otimes 1 \otimes \gamma); (g \otimes 1).$$

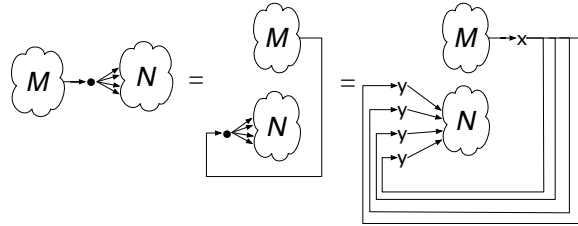
The flat, nominal syntax on the other hand requires the addition just a new line to correspond to the new wire. As before, the flat syntax is more robust to change. However, we can use the link syntax to deal with comonoid structures just by tweaking the type system. To reflect the fact that wire nodes have now exactly one incoming and zero or more outgoing edges we allow contraction and weakening for input variables while output variables preserve the linear discipline.

$$\begin{array}{c}
\frac{}{\Gamma, x \mid - \vdash x : 0 \rightarrow 1} \quad \frac{}{\Gamma \mid x \vdash x : 1 \rightarrow 0} \quad \frac{}{\Gamma \mid - \vdash 1 : 1 \rightarrow 1} \quad \frac{}{\Gamma \mid - \vdash \gamma : 2 \rightarrow 2} \\
\frac{}{\Gamma \mid - \vdash \psi : 1 \rightarrow 2} \quad \frac{}{\Gamma \mid - \vdash \epsilon : 1 \rightarrow 0} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma \mid \Delta' \vdash N : n_1 \rightarrow n_2}{\Gamma \mid \Delta \uplus \Delta' \vdash M; N : m_1 \rightarrow n_2} m_2 = n_1 \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma \mid \Delta' \vdash N : n_1 \rightarrow n_2}{\Gamma \mid \Delta \uplus \Delta' \vdash M \otimes N : m_1 \rightarrow n_2} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad i = 0, 1}{\Gamma \mid \Delta \vdash \text{Tr}^i(M) : m_1 - i \rightarrow m_2 - i} \quad \frac{\Gamma, x \mid \Delta, y \vdash M : m_1 \rightarrow m_2}{\Gamma \mid \Delta \vdash \overline{xy}.M : m_1 \rightarrow m_2}
\end{array}$$

Even though the semantics of link remains the same, the graph invariants are different because of the presence of the co-multiplication and the co-unit. The diagram for  $\overline{xy}.M$  is still the connection of (the single) occurrence of the node labelled with  $x$  to the (zero or more) occurrences of the node(s) labelled with  $y$ , followed by the removal of the labels.

**Lemma 11** (Biflow comonoid definability). *Any biflow diagram with a comonoid structure is definable in terms of composition, tensor and link.*

*Proof.* The proof is again the same as for Lem. 4, with the key graph invariant being that each wire node has one incoming and several outgoing edges. We first reduce all wire nodes so that there exists exactly one wire node between any two constant morphisms. Then we rearrange the diagram so that all constant morphisms are globally set in parallel. Finally, we replace all wire nodes with two fresh variables  $x, y$  and link them.



The linear use of  $x$  and the possibly non-linear use of  $y$  ensures that the top-level *link* construct is typed correctly.  $\square$

From this it follows immediately that the *link* construct is enough in terms of expressiveness, making the explicit use of the co-unit and the co-multiplication optional.

**Theorem 12.** *For any term  $M$  in the language of biflow diagrams with comonoids, there are forms  $\tilde{M}$  and  $\hat{M}$  such that  $\tilde{M}$  is link-free and  $\hat{M}$  is free of structural combinators (identity, symmetry, trace, comultiplication, counit) and  $\llbracket M \rrbracket = \llbracket \tilde{M} \rrbracket = \llbracket \hat{M} \rrbracket$ .*

$$\begin{array}{c}
\frac{}{\Gamma, x \mid \Delta \vdash x : 0 \rightarrow 1} \quad \frac{}{\Gamma \mid x, \Delta \vdash x : 1 \rightarrow 0} \quad \frac{}{\Gamma \mid \Delta \vdash 1 : 1 \rightarrow 1} \\
\frac{}{\Gamma \mid \Delta \vdash \gamma : 2 \rightarrow 2} \quad \frac{}{\Gamma \mid \Delta \vdash \psi : 1 \rightarrow 2} \quad \frac{}{\Gamma \mid \Delta \vdash \epsilon : 1 \rightarrow 0} \\
\frac{}{\Gamma \mid \Delta \vdash \phi : 2 \rightarrow 1} \quad \frac{}{\Gamma \mid \Delta \vdash \eta : 0 \rightarrow 1} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma \mid \Delta \vdash N : n_1 \rightarrow n_2}{\Gamma \mid \Delta \vdash M; N : m_1 \rightarrow n_2} m_2 = n_1 \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad \Gamma \mid \Delta \vdash N : n_1 \rightarrow n_2}{\Gamma \mid \Delta \vdash M \otimes N : m_1 \rightarrow n_2} \\
\frac{\Gamma \mid \Delta \vdash M : m_1 \rightarrow m_2 \quad i = 0, 1}{\Gamma \mid \Delta \vdash \text{Tr}^i(M) : m_1 - i \rightarrow m_2 - i} \quad \frac{\Gamma, x \mid \Delta, y \vdash M : m_1 \rightarrow m_2}{\Gamma \mid \Delta \vdash \overline{xy}.M : m_1 \rightarrow m_2}
\end{array}$$

Figure 7: Type system for spider diagrams

The equational theory of the diagrammatic language is extended by the following two new equations:

$$\boxed{\epsilon = \overline{xy}.x \quad \psi = \overline{xy}.x \otimes y \otimes y.}$$

**Theorem 13.** *The equational theory of biflow diagrams with a comonoid structure is sound and complete.*

*Proof sketch.* The proof follows the same pattern as before. We broaden the scope of the *link* binder then we reason about the diagram in which variables are left uninterpreted. We manipulate the diagram to bring it to the same form as in Lem. 11 so that links can be replaced by a combination of comultiplication and trace. The resulting binder-free diagrams must be equal equationally in the SMT theory.  $\square$

The development of a link syntax and equational theory for a monoid structure follows obvious analog lines.

## 4.2 Spider diagrams

The most flexible diagrammatic syntax combines the monoid and the comonoid using the Frobenius and special axioms, leading to the so-called *spider diagrams*. In the wire homeomorphism model we are using to define equivalent diagrams these axioms are immediately validated. We emphasise that this combination of monoid and comonoid structures must be dictated by the desired application domain. This is just the simplest and most natural way to combine them for a diagrammatic semantics [4].

Besides the new constants  $\phi$  and  $\eta$ , the monoid multiplication and unit, the type judgements now lose all linearity, with link nodes useable zero, one or more times both as input and output (Fig. 7).

Unsurprisingly, the same mathematical properties hold, and the proofs are again slight adaptations of the proofs of the previous sections.

**Theorem 14.** *For any term  $M$  in the language of biflow spider diagrams, there are forms  $\tilde{M}$  and  $\hat{M}$  such that  $\tilde{M}$  is link-free and  $\hat{M}$  is free of structural combinators (identity, symmetry, trace, (co)multiplication, (co)unit) and  $\llbracket M \rrbracket = \llbracket \tilde{M} \rrbracket = \llbracket \hat{M} \rrbracket$ .*

The two equations for the monoid structure are:

$$\boxed{\eta = \overline{xy}.y, \quad \phi = \overline{xy}.x \otimes x \otimes y.}$$

**Theorem 15.** *The equational theory of biflow spider diagrams is sound and complete.*

The proof is again similar, using the fact that the spider diagram is a canonical form which can be reconstructed using the monoid and the comonoid (co) multiplications.

Note that the interaction of the monoid and co-monoid is not necessarily subject to the Frobenius law in diagrammatic languages. This is not a structure that is well studied in its own right, but it turns out to occur naturally in the treatment of digital circuits [9]. The syntax in this case is less natural and the equational properties are more elusive, but the expressiveness and convenience of the notation remains an important attribute.

## 5 Discussion and examples

Comparing syntaxes is difficult, as personal preference and taste will always play a significant role. The conventional syntax, which is flat and nominal, may be rejected out of hand of those who favour structure over everything. Edgar Dijkstra’s *Go To Statement Considered Harmful* letter [6] is a famous (or perhaps infamous) example of rejecting unstructured syntax while simultaneously acknowledging its superior expressiveness:

The **go to** statement as it stands is just too primitive; it is too much an invitation to amek a mess of one’s program. One can regard and appreciate the clauses considered<sup>3</sup> as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

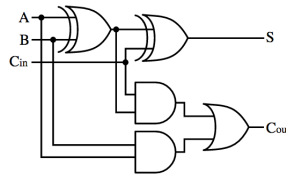
We shall not aim to reopen an old and thorny debate but we will take it as a matter of fact that particularly in the case of diagrammatic notations the unstructured syntax (wires are indeed very much *goto*-like) is dominant.

The methodological advantages of structured syntax are well known and we shall not insist on them too much. When two diagrams have naturally matching interfaces which compose on the nose then the structured syntax can entirely eliminate the need for port and wire names, for the benefit of concision and clarity, as illustrated by the introductory example. Moreover, in the case of uniflow diagrams, which correspond to feedback-free circuits, the type system can be used to prevent unwanted feedback, noting that in the case of certain circuits (e.g. combinational circuits) the introduction of feedback loops can pose serious problems.

The nominal syntax has several advantages which we shall illustrate below. When the diagram itself is unstructured the structured syntax is not all that helpful. Also, when small changes need to be made to a structured diagram they are often algorithmically trivial using nominal syntax, whereas the structured approach can be rather heavy going. Finally, line forking and joining used as combinators are almost always more awkward than the use of names.

### 5.1 Adder

One of the most common digital circuits is the *adder*. It is composed of a *one-bit adder*:



which can be *cascaded* into an arbitrarily long *ripple-carry adder*.

<sup>3</sup>The clauses of structured imperative programming, such as **if-then else**, **while-do**, etc.

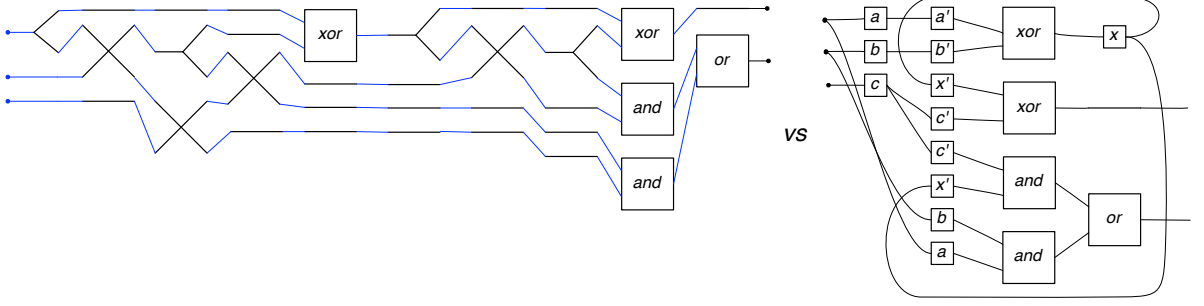
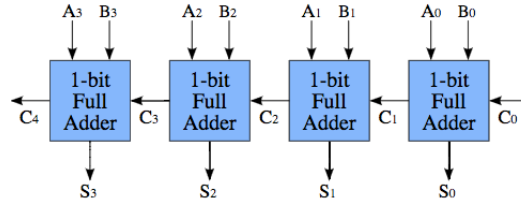


Figure 8: Diagrammatic reconstruction of the structured vs. unstructured representations of a one-bit full adder



Let us use *fork* ( $f$ ), *join* ( $j$ ), *and*, *or*, and *xor* as structural morphisms and constants. A fully structural description of the one-bit adder is:

$$FA = (f \otimes 2); (1 \otimes \gamma \otimes 1); (1 \otimes f \otimes \gamma); (2 \otimes \gamma \otimes 1); (xor \otimes 3); \\ (f \otimes 3); (1 \otimes \gamma \otimes 2); (1 \otimes f \otimes 3); (xor \otimes and \otimes and); (1 \otimes or).$$

The nominal description of the circuit on the other hand is, using biflow spider diagrams:

$$FA = \overline{aa'}. \overline{bb'}. \overline{cc'}. \overline{xx'}. (a \otimes b \otimes c); (a' \otimes b' \otimes x' \otimes c' \otimes c' \otimes x' \otimes b \otimes a); \\ (xor \otimes xor \otimes and \otimes and); (x \otimes 1 \otimes or).$$

This is slightly more verbose but it relies on a conceptually simpler model of the diagram. Instead of forking, joining and crossing wires, each with their own diagrammatic semantics, only variable names need to be used. Two drawings of the circuit which emphasise to the structured, respectively the unstructured, views of the same diagram are given in Fig. 8.

Since the one-bit adder is an unstructured circuit, the structural description above is awkward and artificial. On the other hand, the ripple-carry adder is, in a structured description as elegant and clear as its diagram:

$$RCA = (FA \otimes 6); (1 \otimes FA \otimes 4); (2 \otimes FA \otimes 2); (3 \otimes FA).$$

## 5.2 Instrumented adder

Hardware circuits, much like software, must sometimes be debugged. Debugging can be done in simulation, but a more interesting situation is the case of field-programmable gate arrays (FPGA), which can be debugged during normal execution. In order to do that, a “probe” is placed on the circuit connecting any point in the debugged circuit to a special output port that can be observed during execution. The need to observe, i.e. to make into an output, some arbitrary point in a circuit occurs in many other contexts, not just debugging. Any point in the circuit represents an intermediate computation and it often turns out to be the case that this intermediate computation is needed somewhere else in the circuit, and we often want to avoid recomputing it.

Supposed that we want to observe the output of the first *xor* gate of the second one-bit adder of the full adder. The structural description of the instrumented one-bit adder is:

$$FA' = (f \otimes 2); (i \otimes \gamma \otimes i); (i \otimes f \otimes \gamma); (2 \otimes \gamma \otimes i); (xor \otimes 3); (\textcolor{red}{f} \otimes 3); (\textcolor{red}{1} \otimes f \otimes 3); (2 \otimes \gamma \otimes 2); \\ (\textcolor{red}{2} \otimes f \otimes 3); (\textcolor{red}{1} \otimes xor \otimes and \otimes and); (\textcolor{red}{2} \otimes or).$$

The modified code is in red. The first observation is that in order to propagate a *local* change, *global* modifications are required. Moreover, the modifications require a rather unnatural view of the diagram through the combinators. The change needs to be further propagated in the ripple-carry adder:

$$RCA' = (FA \otimes 6); (1 \otimes FA' \otimes 4); (\textcolor{red}{3} \otimes FA \otimes 2); (\textcolor{red}{4} \otimes FA).$$

In the case of the nominal representation the only required change to the one-bit adder is connecting the variable  $x'$  to the interface:

$$FA = \overline{aa'}. \overline{bb'}. \overline{cc'}. \overline{xx'}. (a \otimes b \otimes c); (a' \otimes b' \otimes x' \otimes c' \otimes c' \otimes x' \otimes b \otimes a); \\ (xor \otimes xor \otimes and \otimes and); (x \otimes \textcolor{red}{x'} \otimes 1 \otimes or),$$

after which the ripple-carry adder is instrumented as above.

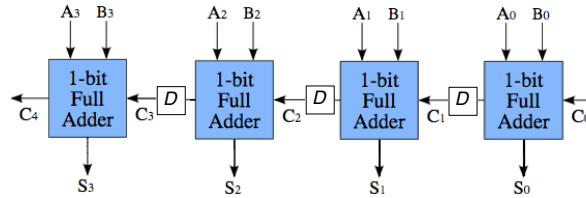
An alternative way to handle this instrumentation is to bring the variables  $x, x'$  into *global* scope. The instrumentation of  $FA''$  consists of removing the binder  $\overline{xx'}$ . The alternative instrumented ripple-carry adder is:

$$RCA'' = \overline{\textcolor{red}{xx'}}. (FA \otimes 6); (1 \otimes FA'' \otimes 4); (2 \otimes FA \otimes 2); (3 \otimes FA \otimes \textcolor{red}{x'}).$$

This solution also takes advantage of the scoping rules for the link, and is perhaps the most attractive.

### 5.3 Clock and reset lines

Supposed now that we require a pipelined version of the ripple-carry adder, which is achieved by placing a delay (D-flip-flop) on each carry line:



The structural solution is rather elegant:

$$RCA = (FA; (\textcolor{red}{1} \otimes D) \otimes 6); (1 \otimes FA; (\textcolor{red}{1} \otimes D) \otimes 4); (2 \otimes FA; (\textcolor{red}{1} \otimes D) \otimes 2); (3 \otimes FA).$$

However, sequential (clocked) circuits require clock lines for synchronisation, and they also require reset lines for proper initialisations. These are *global* nets, which need to be specified. Consider the clock lines only. The D flip-flop has a special input  $ck$  which must be connected with all the other  $ck$  inputs of the other flip-flops. Note that the clock net must be specified, because it is quite possible for a circuit to use multiple clocks.

Structurally, the clocked circuit is:

$$RCA = ((FA \otimes f); (1 \otimes D \otimes ((1 \otimes \gamma); (\gamma \otimes 1)))) \otimes 6); \\ (1 \otimes (FA \otimes f); (1 \otimes D \otimes ((1 \otimes \gamma); (\gamma \otimes 1)))) \otimes 4); \\ (2 \otimes (1 \otimes (FA \otimes f); (1 \otimes D \otimes ((1 \otimes \gamma); (\gamma \otimes 1)))) \otimes 2); \\ (3 \otimes FA).$$

Propagating the clock line via explicit forking and crossing of wires is very complicated and in fact ends up *obscuring* the structure of the circuit. By contrast, using links the clock boilerplate is kept to a minimum and the circuit structure is clearer:

$$RCA = \overline{ck} \overline{ck'}. ck \otimes (FA; (1 \otimes ck'; D) \otimes 6); (1 \otimes FA; (1 \otimes ck'; D) \otimes 4);$$



$$(2 \otimes FA; (1 \otimes ck'; D) \otimes 2); (3 \otimes FA).$$

Reset lines are:

$$RCA = \overline{ck} \overline{ck'} . \overline{rst} \overline{rst'} . ck \otimes rst \otimes (FA; (1 \otimes (ck' \otimes rst'); D) \otimes 6); (1 \otimes FA; (1 \otimes (ck' \otimes rst'); D) \otimes 4); \\ (2 \otimes FA; (1 \otimes (ck' \otimes rst'); D) \otimes 2); (3 \otimes FA).$$

A structural description of the circuit with reset lines is too complicated to write here.

## 6 Discussion, related and further work

Computer programs have always been seen primarily as syntactic artefacts. Code is stored as long sequences of ASCII characters, most often down to the faithful representation of whitespace – *space vs. tab* are amusing but sadly real debates. However, influential designers such as Bret Victor have long argued that code should not necessarily be viewed as text, but diagrammatically<sup>4</sup>, as this representation is much more appealing to human cognition. The switch from textual to diagrammatic notation could be as much of an improvement as the switch from numeric (binary or hexadecimal) to textual notation. And, indeed, diagrammatic representation of *code* exist in some domain-specific areas such as hardware schematics (VHDL, VERILOG) or system representation languages (SIMULINK). But even diagrammatic languages ultimately need a syntactic representation. Current languages for diagrams are low-level, requiring a large number of names, and almost devoid of compositional structure. Our paper aims to improve this, reconciling a style of syntax emerging from the categorical study of diagrams with the more established style.

We have shown how variables can be used within a structural framework for diagram syntax. We showed how from the point of view of expressiveness the structural and the nominal syntax can be equivalent and complementary, if the use of variables is constrained by a suitable type system. We have highlighted a neat correspondence between non-linearity and the presence of a monoid or co-monoid structure in the diagram. By extending the structural notation with the new *link* construct we showed that the existing equational properties are preserved and, moreover, this new structure itself has good equational properties.

We believe that the combination of structural and nominal syntax can greatly improve the readability and usability of diagram languages. Although this paper is exclusively concerned with syntax, it is part of a larger research effort aimed at creating hardware languages more similar to programming languages in terms of their categorical [8] and operational semantics [9]. As further work we intend to create *structural* conservative extensions of diagram languages such as VHDL, VERILOG, SIMULINK or DOT.

Several theoretical aspects we have not studied, but remain subject for further work. The most intriguing perhaps is the categorical semantics for the *link* structure itself. Similar diagrammatic structures have been conjectured in the study of higher-order  $\pi$ -calculus, with the input and output links modelled as adjoint profunctors and a link-like operation modelled as their composition [17]. A smaller issue, but practically important, is understanding the syntax and the axiomatisation of links for diagrams with monoid and comonoid structures in the absence of the Frobenius law. These structures occur in diagrams modelling systems with directed flow of information, such as digital circuits, which are well motivated by applications. Finally, the various definability results (e.g. Lem. 4) suggest the possibility of arriving at normal forms for various classes of diagrams.

In terms of the graph semantics, the frame point graphs is not the only possible starting point. Cospans of graphs [14] or hypergraphs [3] have also been used as concrete examples of diagrammatic monoidal categories. These alternative presentations may arguably offer, to the theorist, more conceptual clarity. What we find attractive about FPGs is the fact that they can be presented in a concrete way, eliding much of the categorical apparatus. We believe this could make our work accessible to a broader audience. Moreover, the *link* construct is particularly easy to interpret in FPGs. However, since our main reference points are the syntax and its equational properties, we believe the choice of concrete representation not to be crucial.

<sup>4</sup><http://worrydream.com/#!/TheFutureOfProgramming>

## References

- [1] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 415–425, 2004.
- [2] P. Bjesse, K. Claessen, M. Sheeran, and S. Singh. Lava: Hardware design in Haskell. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98), Baltimore, Maryland, USA, September 27-29, 1998.*, pages 174–184, 1998.
- [3] F. Bonchi, P. Sobocinski, and F. Zanasi. Full abstraction for signal flow graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 515–526, 2015.
- [4] B. Coecke and R. Duncan. Interacting quantum observables. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pages 298–310, 2008.
- [5] V. E. Căzănescu and G. Stefănescu. Towards a new algebraic foundation of flowchart scheme theory. *Fundam. Inf.*, 13(2):171–210, May 1990.
- [6] E. W. Dijkstra. Letters to the editor: go to statement considered harmful. *Commun. ACM*, 11(3):147–148, 1968.
- [7] D. R. Ghica. Diagrammatic reasoning for delay-insensitive asynchronous circuits. In *Computation, Logic, Games, and Quantum Foundations*, pages 52–68. Springer, 2013.
- [8] D. R. Ghica and A. Jung. Categorical semantics of digital circuits. In R. Piskac and M. Talupur, editors, *Formal Methods in Computer-Aided Design (FMCAD), 2016, Mountain View, California, USA*.
- [9] D. R. Ghica, A. Jung, and A. Lopez. Diagrammatic operational semantics for digital circuits. (submitted to LICS).
- [10] P. Hackney and M. Robertson. On the category of props. *Applied Categorical Structures*, 23(4):543–573, 2015.
- [11] M. Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, pages 196–213, 1997.
- [12] A. Joyal and R. Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991.
- [13] A. Kissinger. Pictures of processes: automated graph rewriting for monoidal categories and applications to quantum computing. *arXiv preprint arXiv:1203.0202*, 2012.
- [14] R. Rosebrugh, N. Sabadini, and R. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15(6):164–177, 2005.
- [15] M. Sadrzadeh, S. Clark, and B. Coecke. The Frobenius anatomy of word meanings I: subject and object relative pronouns. *J. Log. Comput.*, 23(6):1293–1317, 2013.
- [16] P. Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
- [17] J. Vicary. personal communication.
- [18] J. Vicary, A. Kissinger, and K. Bar. GLOBULAR: An online proof assistant for higher-dimensional rewriting. <http://globular.science>, 2016.